ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

ISTI – CNR

Internet Services Technology Center

Guida all'installazione di Java e JBoss in ambiente OpenVMS

Pisa, 26 Marzo 2007

Table of Contents

1. Ambiente di sviluppo

In questa sezione viene descritto l'ambiente di lavoro hardware e software utilizzato per effettuare le attività di verifica e collaudo dell'Application Server JBoss e della Java Virtual Machine (JVM), su sistema operativo OpenVMS I64 v 8.3.

1.1 Il server ed il sistema operativo

Si tratta di un server HP Integrity RX2620 con sistema operativo HP OpenVMS I64 versione 8.3, aggiornata con le ultime "*Mandatory Patches*" rilasciate fino al 18 Dicembre 2006, in base a quanto espressamente raccomandato dalle specifiche di installazione della JDK 5.0-2 su Itanium, contenute nella "*Release Notes*", consultabile al sito:

http://h18012.www1.hp.com/java/documentation/1.5.0/ivms/docs/release_notes.html

Le "*Mandatory Patches*" possono essere scaricate dal sito:

http://h71000.www7.hp.com/serv_support.html

2 . JDK 5.0-2

Si tratta del "*J2SE™ Development Kit (JDK) 5.0-2*", per il Sistema Operativo OpenVMS I64 v 8.2-1 o superiori, per la piattaforma JAVA™, scaricabile dal sito:

http://h18012.www1.hp.com/java/download/ivms/1.5.0/index.html

Questo Kit include il "*Java Runtime Environment*" (JRE), che fornisce le librerie, la "*Java Virtual Machine*" (JVM) ed altre componenti necessarie per eseguire applet ed applicazioni scritte in linguaggio Java ed in più, offre compilatori e debugger necessari per sviluppare applet e programmi Java, in ambiente OpenVMS.

Il kit JDK 5.0-2 implementa la J2SE™ 5.0 e si basa sulla release di Solaris per la J2SE 1.5.0_10 della SUN. Il kit contiene una "*HotSpot Virtual Machine*", che utilizza la tecnologia di compilazione "*Just-In-Time*" (JIT), progettata per ottenere la massima velocità nell'esecuzione di applicazioni sviluppate in ambiente "server side". La "*HotSpot VMS*" funziona solo su sistemi OpenVMS I64 e non anche su piattaforma OpenVMS Alpha.

Il kit JDK 5.0-2 è compatibile con le precedenti versioni SDK 1.4.x, a meno delle incompatibilità espressamente descritte nel documento della Sun "*Java 2 Platform Compatibility with Previous Releases*", consultabile al sito:

http://java.sun.com/javase/technologies/compatibility.jsp

2.1 Problemi e particolarità su OpenVMS I64 v 8.3

Nel documento "*Release Notes J2SE™ Development Kit (JDK) 5.0-2 for the OpenVMS I64 Operating System for the Java™ Platform*", sono contenute le informazioni utili riguardo alle

caratteristiche nuove e conosciute di questo Kit ed alle modalità di installazione in ambiente OpenVMS I64. Tale documento è consultabile al sito:

http://h18012.www1.hp.com/java/documentation/1.5.0/ivms/docs/release_notes.html

Nella sezione "*Prerequisites*" è espressamente richiesto che il JDK 5.0 sia installato su un disco formattato ODS-5, che garantisce la compatibilità con il mondo Unix, permettendo ad esempio, di utilizzare nomi di file con più punti o avere fino a 256 livelli di directory.

Nella sezione "*Know Issues*" contenuta nel documento, sono specificate le limitazioni della versione JDK 5.0-2 per il sistema operativo OpenVMS I64, come ad esempio:

- -Xeprof, l'opzione per la generazione dei dati per HPJmeter, non è supportata.
- Il supporto "*Java Networking IPV6*" non è incluso.
- I tool jinfo, jmap, jsadebugd e jstack, introdotti nella JDK 5.0, non sono supportati in OpenVMS I64.
- -XX:+UseParallelOldGC, tale opzione non è supportata.
- Java non funziona correttamente se sono definiti i seguenti "*nomi logici*":
  - DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION
  - DECC$FILENAME_UNIX_ONLY

  Note: Java potrebbe non funzionare correttamente con la definizione di altri nomi logici di tipo DECC$*.

In questo documento, non è specificato che il "*JDK 5.0-2*" per i sistemi OpenVMS, non prevede il "*JavaBeans Packager*" Tool, previsto invece, per i sistemi UNIX e Windows.

Nel documento "*User Guide: J2SE™ Development Kit (JDK) 5.0x for the OpenVMS Operating System for the Java™ Platform*", consultabile al sito:

http://h18012.www1.hp.com/java/documentation/1.5.0/ovms/docs/user_guide.html

sono contenute le informazioni necessarie per utilizzare il "*J2SE™ Development Kit (JDK) 5.0-x*", sul sistema operativo OpenVMS, dove "x" sta ad indicare il numero di versione della release che si sta utilizzando, in questo caso, la "2". La versione installata può essere visualizzata digitando il comando "java -version", dopo aver effettuato il setup dell'ambiente.

In particolare, nella "*User Guide*" vengono specificate le differenze che esistono nello sviluppo di applicazioni Java, in ambiente OpenVMS, rispetto ad altri ambienti, come quello Unix e la possibilità di rendere i due mondi compatibili, utilizzando ad esempio, un disco formattato ODS-5, definendo alcuni nomi logici di tipo DECC$* e JAVA$* ed attivando alcuni algoritmi di mappatura dei nomi.

Nella sezione "*Interpreting Commands and OpenVMS Operating System Differences*", vengono evidenziate alcune differenze nell'esecuzione dei comandi in ambiente OpenVMS, rispetto al sistema Unix, come ad esempio, la diversa modalità di definizione del "JAVA$CLASSPATH" o il fatto che OpenVMS non è "*case-sensitive*", mentre Java si.

Inoltre, il JRE ("*Java Runtime Environment*") insiste sulla relazione uno-ad-uno tra il nome della

classe ed il nome del file. Vale a dire, che il file deve essere chiamato con lo stesso nome della classe "*main*", rispettando anche il "*case-sensitive*". Perciò, se la classe viene definita ad esempio:

```
public class CaseTest {
.............
}
```

allora, anche il file ".java" deve essere chiamato nello stesso modo, cioè "Case*Test.java*".

2.2 Configurazione dell'ambiente JDK 5.0-2

Una volta installato il Kit JDK 5.0-2, per poterlo utilizzare correttamente nel sistema OpenVMS I64 v 8.3, è necessario effettuare una serie di operazioni:

1. La definizione dei "nomi logici" che controllano alcune delle caratteristiche speciali della "C Run Time Library" (C RTL), come ad esempio, la conservazione del "*case"* nella riga di comando, negli argomenti e nel nome dei file:

   - ✔ define DECC$EFS_CHARSET enable
   - ✔ define DECC$ARGV_PARSE_STYLE enable
   - ✔ define DECC$EFS_CASE_PRESERVE enable

   - ◆ Alcuni applicativi che lavorano in ambiente Java, come ad esempio, l'Application Server "*JBoss*", potrebbero richiedere la definizione di ulteriori variabili logiche, per funzionare correttamente, come vedremo in seguito.

2. La definizione degli algoritmi di "*Filename Mapping*" tra il mondo Unix e quello VMS:

   - ✔ a questo proposito, si è scelto di favorire il risparmio nell'utilizzo della memoria e quindi, ridurre al minimo la mappatura dei nomi, abilitando soltanto l'algoritmo:

     ```
     define/job  JAVA$FILENAME_CONTROLS  8
     ```

3. Il "*set up*" dell'ambiente di Java, con il comando:

   ```
   @SYS$COMMON:[JAVA$150.COM]JAVA$150_SETUP.COM
   ```

   Il file "JAVA$150_SETUP.COM", esegue una serie di operazioni per la configurazione dell'ambiente Java, tra cui, ad esempio:

   - ✔ la definizione di simboli che consentono l'utilizzo dei comandi Java (*java*, *javac*, *jar*, etc....) come comandi esterni;
   - ✔ la definizione di nomi logici per le "*shareable images*", in base alla tecnologia *Virtual Machine* abilitata; per I64, il default è "*Hotspot Virtual Machine*";
   - ✔ la definizione del valore per il nome logico "JAVA$FILENAME_CONTROLS", che determina gli algoritmi di mappatura dei nomi tra il mondo Unix e quello VMS;

- ✔ a tal proposito, si è deciso di disabilitare la mappatura di default (che abilita"*TUTTI"* gli algoritmi indicati nel file "java$filename_controls.com"), per limitare l'utilizzo di memoria, ed abilitare soltanto l'algoritmo indicato precedentemente ("JAVA$M_UNIX_AND_VMS").

  Ad ogni modo, ciascun algoritmo specificato nel file "java$filename_controls.com", può essere abilitato all'occorrenza, andando a togliere i relativi commenti all'interno di questo file.

4. La definizione dei nomi logici "JAVA_HOME" e "JAVA$CLASSPATH":

   - ✔ DEFINE/SYSTEM/EXECUTIVE/TRANSLATION=CONCEALED JAVA_HOME - SYSCOMMON:[JAVA$150.]
   - ✔ DEFINE/SYSTEM/EXECUTIVE/TRANSLATION=CONCEALED JAVA$CLASSPATH [], - JAVA_HOME:[LIB]

3. SDK 1.4.2-3

Si tratta del Software Development Kit (SDK) v 1.2.4-3, per il Sistema Operativo OpenVMS I64 v 8.2-1 e superiori, per la piattaforma JAVA™. Il Kit si basa sulla Reference Release di Solaris v 1.4.2_09 ed implementa la Sun Microsystem's JAVA™ 2 SDK, Standard Edition (J2SDK).

Tale versione SDK 1.4.2-3 di Java può essere sostituita con quella più recente JDK 5.0-2, considerando la forte compatibilità garantita dalla Sun tra le due versioni, a meno di quanto espressamente indicato, in casi particolari, nel documento consultabile al sito:

http://java.sun.com/javase/technologies/compatibility.jsp

Il Kit SDK 1.4.2-3 è stato installato e configurato in ambiente OpenVMS I64 v 8.3.

3.1 Problemi e particolarità su OpenVMS I64 v 8.3

Nel documento "*Release Notes Software Development Kit (SDK) v 1.4.2-3 for the OpenVMS Integrity (I64) Operating System for the Java™ Platform*", sono contenute le informazioni utili riguardo alle caratteristiche nuove e conosciute di questo Kit ed alle modalità di installazione in ambiente OpenVMS I64. Tale documento è consultabile al sito:

http://h18012.www1.hp.com/java/documentation/1.4.2/ivms/docs/release_notes.html

Nel documento "*User Guide Software Development Kit (SDK) v 1.4.x for the OpenVMS Operating System for the Java™ Platform*", consultabile al sito:

http://h18012.www1.hp.com/java/documentation/1.4.2/ovms/docs/user_guide.html

sono contenute le informazioni necessarie per utilizzare il "Software *Development Kit (JDK) v 1.4.x*", sul sistema operativo OpenVMS, dove "x" sta ad indicare il numero di versione della release che si sta utilizzando, in questo caso, la "2". La versione installata può essere visualizzata digitando il comando "java -version", dopo aver effettuato il setup dell'ambiente.

3.2 Configurazione dell'ambiente SDK 1.4.2-3

Una volta installato il Kit SDK 1.4.2-3, per poterlo utilizzare correttamente nel sistema OpenVMS I64 v 8.3, è necessario effettuare una serie di operazioni, già elencate nella sezione 1.2.2 riguardo alla "Configurazione dell'ambiente JDK 5.0-2", considerando che è necessario sostituire la versione "150" di Java, con quella "142".

4. JBOSS 4.0.5.GA

Si tratta della Release "jboss-4.0.5-GA.zip" scaricabile dal sito:

http://labs.jboss.com/portal/jbossas/download

> ***[Nota]:*** Assicurarsi di utilizzare un applicativo di "unzip" ODS-5 compatibile, per decompattare il Kit. A tal proposito, abbiamo utilizzato "UnZip 6.00c BETA", scaricabile dal sito:
>
> ftp://ftp.info-zip.org/pub/infozip/
>
> o dal sito:
>
> http://antinode.org/ftp/info-zip/
>
> Tale Kit deve essere compilato, mentre una versione eseguibile, che abbiamo ottenuto compilando il Kit su una macchina OpenVMS I64 v8.3, può essere scaricata dal sito:
>
> http://mx.isti.cnr.it/unzip/

"JBOSS", acronimo di "***J****ava* ***B****ean* ***O****pen* ***S****ource* ***S****erver*", è un Application Server, implementato in Java, compatibile con le specifiche della piattaforma "*Java 2 Enterprise Edition*" (J2EE™) v 1.4 della SUN e quindi, in grado di sfruttarne tutte le potenzialità.

Sul sito http://labs.jboss.com/portal/jbossas/docs è possibile consultare la documentazione per installare JBoss sulla maggior parte dei sistemi operativi (Windows, Unix, MacOS e Solaris), mentre non esiste ancora, alcuna documentazione riguardo all'installazione di JBoss sul sistema operativo OpenVMS.

Nel successivo paragrafo verrà descritta la procedura di installazione dell'Application Server *JBoss*, effettuato dall'ISTI-CNR, sulla piattaforma Itanium (I64), equipaggiata con il sistema operativo OpenVMS v 8.3.

JBoss è stato eseguito sia in ambiente "*Java 5*" che "*Java 1.4*".

4.1 Installazione su OpenVMS I64 v 8.3

L'installazione di JBoss ha richiesto l'esecuzione di una serie di operazioni:

1. Prima di tutto, è necessario creare un utente non privilegiato sotto cui eseguire l'Application

2. Server, ad esempio, l'utente "JBOSS", che avrà l'*ownership* della directory di installazione di "JBoss".

   Una volta creato l'utente "JBOSS", si accederà con i suoi privilegi al sistema OpenVMS e si procederà a decomprimere il kit "*jboss-4.0.5-GA.zip*" nella directory di "sys$login" dell'utente (notare che tale utente sarà il "*proprietario*" della directory di JBoss).

   Noi abbiamo creato l'utente "JBOSS" con una serie di diritti e caratteristiche, ma queste possono essere modificate in ogni momento, a seconda delle necessità. Infatti, è probabile che alcuni valori, come ad esempio, "Wsdef", "Wsquo", "Pgflquo", dovranno essere aumentati. Ma considerazioni più precise a riguardo, potranno essere fatte nel momento in cui l'Application Server lavorerà "*a pieno regime*".

   Attualmente, la configurazione dell'utente "JBoss", sul nostro sistema è la seguente:

Maxjobs:	0	Fillm:	4096	Bytlm:	1000000
Maxacctjobs:	0	Shrfillm:	0	Pbytlm:	0
Maxdetach:	0	BIOlm:	200	JTquota:	16000
Prclm:	10	DIOlm:	200	WSdef:	32000
Prio:	4	ASTlm:	300	WSquo:	32000
Queprio:	0	TQElm:	100	WSextent:	128000
CPU:	(none)	Enqlm:	4000	Pgflquo:	2097152

   Authorized Privileges:

   NETMBX TMPMBX

   Default Privileges:

   NETMBX TMPMBX

   Inoltre, ci sono anche altri parametri da prendere in considerazione, che può essere necessario modificare:

   - ✔ "CHANNELCNT" dovrebbe essere settato ad un valore grande almeno quanto il valore del "UAF FILLM". Una buona regola, può essere quella di scegliere il valore più grande tra: il valore corrente, il valore del "UAF FILLM" e 4096. Per esempio, potrebbe essere 8192.

   - ✔ "PQL_DWSQUOTA" e "PQL_MWSQUOTA" potrebbe essere necessario incrementarli fino ad un valore vicino a 600000.

   Inoltre, potrebbe essere molto utile dare un'occhiata anche ad altri parametri di "SYSGEN" di tipo "PQL_xxxx", in quanto l'istanza di "JBoss" potrebbe richiedere dei valori più elevati, per partire.

   Infatti, è stato necessario ad esempio, incrementare, il valore di default del parametro

"PQL_DFILLM", da 128 a 250.

> ***[Nota]:*** Ricordarsi di effettuare la generazione del sistema nel caso che i parametri modificati non siano di tipo dinamico.

Noi siamo arrivati a questa conclusione, osservando il file di "*log*" creato allo start up dell'Application Server, in caso di fallimento, che potrebbe visualizzare un messaggio di errore simile al seguente:

```
17:58:48,606 ERROR [MainDeployer] Could not make local copy for
file:/JBOSS_HOME/server/default/deploy/jbossws14.sar/xmlsec.jar
java.io.FileNotFoundException:
/JBOSS_HOME/server/default/tmp/deploy/tmp58008xmlsec.jar(i/oerror (errno:5))
```

2. Eseguire il login al sistema come utente "JBOSS" e fare l'unzip del kit in una qualsiasi directory di destinazione, di un disco formattato ODS-5, utilizzando un decompressore "ODS-5" compatibile. Questo accorgimento, è necessario in ambiente OpenVMS per generare nomi di file e di directory compatibili con quelli utilizzati dagli altri sistemi.

   > ***[Nota]:*** Per utilizzare i nomi estesi è necessario attivare il relativo meccanismo di parsing, con il comando:
   >
   > SET PROCESS/PARSE_STYLE=EXTENDED

   Ad esempio, la directory JBoss che verrà creata con la decompressione, potrà contenere più punti all'interno del suo nome: "*jboss-4^.0^.5^.GA*".

3. La release "*jboss-4.0.5-GA*" richiede l'installazione e la configurazione dell'ambiente Java, nelle versioni 1.4 o 5.0 della *Java Virtual Machine*. La documentazione di installazione "The JBoss 4 Installation Guide JBoss AS 4.0.4 Release 1 (2006)", consultabile al sito:

   http://docs.jboss.com/jbossas/guides/installguide/r1/en/html_single/

   raccomanda l'installazione della versione Java 5.0 nel caso un cui si preveda l'utilizzo della nuova tecnologia "*Enterprise Java Beans (EJB) 3.0*".

   In particolare, la configurazione dell'ambiente Java, esposta nel paragrafo "*1.2.2 Configurazione dell'ambiente JDK 5.0-2*", si è dimostrata corretta ed adeguata all'installazione dell'Application Server JBoss 4.0.5.

4. Definizione del nome logico "JBOSS_HOME", che punta alla directory di installazione dell'Application Server:

   - ✔ DEFINE/SYSTEM/EXECUTIVE/TRANSLATION=CONCEALED - JBOSS_HOME DISK$:[jboss-4^.0^.5^.GA .]

   > ***[Nota]:*** la directory "jboss" sopra, è la directory di root dell'utente "JBOSS", in cui il Kit è stato decompresso.

5. Si è resa necessaria la creazione di un file di "*start up*" di JBoss, "*jboss_startup.com"*, contenente le seguenti istruzioni:

```
run/detach/process_name="JBoss Server" -
/input=JBOSS_HOME:[000000]jboss_command.com -
/output=JBOSS_HOME:[000000]jboss_output.log -
sys$system:loginout.exe
```

e da lanciare come processo "*detached*" per l'utente "JBOSS".

Il file "*jboss_command.com*" contiene tutte le configurazioni necessarie per un corretto start up dell'Application Server. Tale script DCL svolge lo stesso compito del "*run.sh"*, utilizzato dal sistema Unix per lanciare l'application server.

All'interno di questo file, vengono definiti una serie di nomi logici per garantire la compatibilità con il mondo Unix e Java ed il comando Java, per eseguire il "run" di JBoss:

```
define    DECC$EXEC_FILEATTR_INHERITANCE          enable
define    DECC$FILE_OWNER_UNIX                    enable
define    DECC$FILE_PERMISSION_UNIX               enable
define    DECC$FILE_SHARING                       enable
define    DECC$POSIX_SEEK_STREAM_FILE             enable
define    DECC$PIPE_BUFFER_SIZE                   32768

define    JAVA$ENABLE_ENVIRONMENT_EXPANSION       true
define    JAVA$CREATE_DIR_WITH_OWNER_DELETE       true
define    JAVA$DELETE_ALL_VERSIONS                true
define    JAVA$RENAME_ALL_VERSIONS                true
```

```
java "-Xms128m" "-Xmx128m" "-Dprogram.name=jb_run.com" "-Djava.
endorsed.dirs=RX2620$DKA0/jboss-4^.0^.5^.GA/lib/endorsed" -
-classpath JAVA_CLASSPATH:/JBOSS_HOME/bin/run.jar "org.jboss.Main" -c default
```

6. Prima di eseguire il file di "start up" di JBoss (*jboss_startup.com*) è necessario effettuare il "set up" dell'ambiente di Java, come specificato nel paragrafo "*1.2.2 Configurazione dell'ambiente JDK 5.0-2*", in cui la gran parte delle configurazioni sono necessarie per rendere compatibili i mondi OpenVMS, Unix e Java.

7. Si è resa necessaria la creazione di un file di "*shutdown*" dell'Application Server. Questo script DCL svolge lo stesso ruolo dello scripti UNIX "*shutdown.sh*", utilizzato per fermare l'Application Server. Questo script contiene il seguente comando Java:

```
java  -classpath  JAVA$CLASSPATH:/JBOSS_HOME/bin/shutdown.jar -
"org.jboss.Shutdown" -S
```

dove, "JAVA$CLASSPATH" punta alla directory di Java che contiene l'archivio "*tools.jar*" (Paragrafo *2.2 Configurazione dell'ambiente JDK 5.0-2*).

4.2 Starting e Stopping dell'Application Server

Una volta che l'Application Server è stato installato, seguendo le istruzioni indicate nei paragrafi precedenti, il passo successivo è quello di mandarlo in esecuzione.

Noi abbiamo provato a lanciare l'Application Server sia come utente "*SYSTEM*" con tutti i suoi privilegi di amministratore, sia come utente non privilegiato "JBOSS", come processo "*detached*", attraverso il comando seguente:

submit/user=jboss/noprint/nolog jboss_startup.com

Per rendere automatica la partenza di JBoss all'avvio del sistema, è necessario inserire il comando specificato sopra, nel file di startup di OpenVMS "SYS$MANAGER:SYSTARTUP_VMS.COM".

Esempio di output di "show system":

Pid	***Process Name***	***State***	***Pri***	***I/O***	***CPU***	***Page flts***	***Pages***
.....							
0000445	JBoss Server	CUR 1	4	35098	0 00:00:46.21	26237	21358 M
.....							

Se si dà un'occhiata al file di "log" (che abbiamo chiamato "*jboss_output.log*"), si dovrebbe vedere qualcosa di simile al messaggio seguente:

```
11:41:38,110 INFO  [Server] Starting JBoss (MX MicroKernel)...
11:41:38,116 INFO  [Server] Release ID: JBoss [Zion] 4.0.5.GA (build: CVSTag=Branch_4_0 date=200610162339)
11:41:38,118 INFO  [Server] Home Dir: /JBOSS_HOME
11:41:38,119 INFO  [Server] Home URL: file:/JBOSS_HOME/
11:41:38,120 INFO  [Server] Patch URL: null
11:41:38,122 INFO  [Server] Server Name: default
11:41:38,122 INFO  [Server] Server Home Dir: /JBOSS_HOME/server/default
11:41:38,123 INFO  [Server] Server Home URL: file:/JBOSS_HOME/server/default/
11:41:38,123 INFO  [Server] Server Log Dir: /JBOSS_HOME/server/default/log
11:41:38,123 INFO  [Server] Server Temp Dir: /JBOSS_HOME/server/default/tmp
11:41:38,124 INFO  [Server] Root Deployment Filename: jboss-service.xml
11:41:39,148 INFO  [ServerInfo] Java version: 1.5.0,Hewlett-Packard Company
11:41:39,154 INFO  [ServerInfo] Java VM: Java HotSpot(TM) Server VM 1.5.0-2 12/02/2006-21:30 IA64,Hewlett-Packard
Company
11:41:39,154 INFO  [ServerInfo] OS-System: OpenVMS V8.3,ia64
11:41:40,134 INFO  [Server] Core system initialized
11:41:44,003 INFO  [WebService] Using RMI server codebase: http://itanium.isti.cnr.it:8083/
11:41:44,093 INFO  [Log4jService$URLWatchTimerTask] Configuring from URL: resource:log4j.xml
11:42:10,883 INFO  [ServiceEndpointManager] WebServices: jbossws-1.0.3.SP1 (date=200609291417)
11:42:17,570 INFO  [Embedded] Catalina naming disabled
11:42:17,692 INFO  [ClusterRuleSetFactory] Unable to find a cluster rule set in the classpath. Will load the default rule set.
11:42:17,694 INFO  [ClusterRuleSetFactory] Unable to find a cluster rule set in the classpath. Will load the default rule set.
11:42:18,457 INFO  [Http11BaseProtocol] Initializing Coyote HTTP/1.1 on http-0.0.0.0-8080
11:42:18,460 INFO  [Catalina] Initialization processed in 766 ms
11:42:18,467 INFO  [StandardService] Starting service jboss.web
11:42:18,475 INFO  [StandardEngine] Starting Servlet Engine: Apache Tomcat/5.5.20
11:42:18,545 INFO  [StandardHost] XML validation disabled
11:42:18,592 INFO  [Catalina] Server startup in 126 ms
11:42:19,032 INFO  [TomcatDeployer] deploy, ctxPath=/invoker, warUrl=.../deploy/http-invoker.sar/invoker.war/
11:42:19,848 INFO  [WebappLoader] Dual registration of jndi stream handler: factory already defined
```

```
11:42:21,050 INFO  [TomcatDeployer] deploy, ctxPath=/, warUrl=.../deploy/jbossweb-tomcat55.sar/ROOT.war/
11:42:22,489 INFO  [TomcatDeployer] deploy, ctxPath=/jbossws, warUrl=.../tmp/deploy/tmp60584jbossws-context-exp.war/
11:42:23,032 INFO  [TomcatDeployer] deploy, ctxPath=/jbossmq-httpil, warUrl=.../deploy/jms/jbossmq-httpil.sar/jbossmq-
httpil.war/
11:42:26,628 INFO  [TomcatDeployer] deploy, ctxPath=/web-console, warUrl=.../deploy/management/console-mgr.sar/web-
console.war/
11:42:28,906 INFO  [MailService] Mail Service bound to java:/Mail
11:42:30,153 INFO  [RARDeployment] Required license terms exist, view META-INF/ra.xml in .../deploy/jboss-ha-local-jdbc.rar
11:42:30,477 INFO  [RARDeployment] Required license terms exist, view META-INF/ra.xml in .../deploy/jboss-ha-xa-jdbc.rar
11:42:30,844 INFO  [RARDeployment] Required license terms exist, view META-INF/ra.xml in .../deploy/jboss-local-jdbc.rar
11:42:31,178 INFO  [RARDeployment] Required license terms exist, view META-INF/ra.xml in .../deploy/jboss-xa-jdbc.rar
11:42:31,566 INFO  [RARDeployment] Required license terms exist, view META-INF/ra.xml in .../deploy/jms/jms-ra.rar
11:42:31,844 INFO  [RARDeployment] Required license terms exist, view META-INF/ra.xml in .../deploy/mail-ra.rar
11:42:34,357 INFO  [WrapperDataSourceService] Bound ConnectionManager
'jboss.jca:service=DataSourceBinding,name=DefaultDS' to JNDI n
11:42:35,210 INFO  [A] Bound to JNDI name: queue/A
11:42:35,213 INFO  [B] Bound to JNDI name: queue/B
11:42:35,216 INFO  [C] Bound to JNDI name: queue/C
11:42:35,217 INFO  [D] Bound to JNDI name: queue/D
11:42:35,219 INFO  [ex] Bound to JNDI name: queue/ex
11:42:35,333 INFO  [testTopic] Bound to JNDI name: topic/testTopic
11:42:35,336 INFO  [securedTopic] Bound to JNDI name: topic/securedTopic
11:42:35,338 INFO  [testDurableTopic] Bound to JNDI name: topic/testDurableTopic
11:42:35,347 INFO  [testQueue] Bound to JNDI name: queue/testQueue
11:42:35,538 INFO  [UILServerILService] JBossMQ UIL service available at : /0.0.0.0:8093
11:42:35,709 INFO  [DLQ] Bound to JNDI name: queue/DLQ
11:42:36,834 INFO  [ConnectionFactoryBindingService] Bound ConnectionManager
'jboss.jca:service=ConnectionFactoryBinding,name=JmsXA'
11:42:39,235 INFO  [TomcatDeployer] deploy, ctxPath=/jmx-console, warUrl=.../deploy/jmx-console.war/
11:42:42,221 INFO  [Http11BaseProtocol] Starting Coyote HTTP/1.1 on http-0.0.0.0-8080
11:42:42,496 INFO  [ChannelSocket] JK: ajp13 listening on /0.0.0.0:8009
11:42:42,551 INFO  [JkMain] Jk running ID=0 time=0/163  config=null
11:42:42,567 INFO  [Server] JBoss (MX MicroKernel) [4.0.5.GA (build: CVSTag=Branch_4_0 date=200610162339)] Started in
1m:4s:438ms
```

Il funzionamento dell'Application Server può essere verificato andando alla pagina del "JBoss Web Server", in esecuzione sulla porta 8080 e digitando su un "Browser" l'indirizzo del server (nome_host):

http://nome_host:8080/

Si può anche navigare attraverso la console "JBoss JMX" (*Java Management eXtension*) digitando sul Browser l'indirizzo:

http://nome_host:8080/jmx-console

Quest'ultima rappresenta la console di gestione di JBoss che fornisce una vista degli "*JMX Mbeans*", i quali costituiscono il Server. Questa console fornisce molte informazioni riguardo all'Application Server che è in esecuzione e consente di modificare, attivare e disattivare alcune configurazioni e componenti.

Per fermare l'Application Server è necessario eseguire lo script DCL di shutdown, descritto nella sezione precedente. Per rendere automatico lo shutdown dell'Application Server allo shutdown del sistema, è necessario inserire questo file all'interno di "SYS$MANAGER:SYSHUTDOWN.COM".

5. Verifica del Comportamento della JVM su OpenVMS

In questa sezione viene descritto il lavoro di verifica del comportamento della "Java Virtual Machine" in ambiente OpenVMS v8.3 I64, equipaggiato con "JDK 5.0".

Il "JDK" per Itanium contiene la "HotSpot Virtual Machine". La tecnologia di compilazione run-time "HotSpot" è stata sviluppata per fornire performance ottimali su sistemi OpenVMS I64.

5.1 Gestione della Memoria

Il JDK 5.0 prevede il "*Generational Garbage Collection*", che come vedremo nell'esecuzione del Test seguente, permette delle ottimizzazioni che consentono di migliorare le performance degli applicativi scritti in linguaggio Java. In particolare, il "*Generational Garbage Collection*" può essere piuttosto veloce rispetto al "*Full Garbage Collection*", in quanto fa uso di due spazi di memoria: "*young space*" per gli oggetti appena creati ed "*old space*" per gli oggetti che hanno subìto la Full GC e sono sopravvisuti. Successivamente e fino a quando lo "*young space*" non è pieno, la GC viene effettuata soltanto nello "*old space*", con un significativo risparmio di tempo.

Il JDK 5.0 prevede una specifica opzione "-Xverbose:gc" da specificare a run-time quando viene lanciato l'applicativo Java, la quale visualizza informazioni dettagliate sullo spazio dello Java Heap, prima e dopo la Garbage Collection.

Grazie all'output di tale opzione, è possibile capire se l'applicativo ha bisogno di una quantità minima garantita di questo Heap, da specificare con l'opzione "-Xms<value>m" (ad esempio, -Xms256m). Per capire quale valore può essere adatto, si può monitorare l'attività di Heap del nostro applicativo, eseguendo l'opzione "-Xverbose:gc".

Se si rileva che viene eseguito un alto numero di GC in uno spazio limitato di tempo, allora può essere necessario incrementare la dimensione dell'Heap al massimo che si può, senza causare eccessivi "page fault".

5.2 Verifica del comportamento della JVM su operazioni "Memory-intensive" e "CPU-intensive" e sull'utilizzo dei "Thread"

In questo paragrafo viene descritto l'utilizzo di un applicativo per testare il comportamento della JVM in caso di operazioni "*Memory-intensive*" e "*CPU-intensive*" e l'utilizzo dei "*Thread*". In altre parole, il sistema viene "*stressato*" ad eseguire solo operazioni sulla memoria o solo operazioni di computazione e poi vengono calcolati i tempi di esecuzione, anche in relazione al numero di thread che vengono utilizzati per eseguire il lavoro.

L'applicativo utilizzato per fare i Test, è reperibile sul sito della Sun, all'indirizzo:

http://java.sun.com/developer/technicalArticles/Programming/JVMPerf/

con la possibilità di utilizzarlo liberamente, come espressamente indicato dalla licenza di utilizzo.

5.2.1 Test Program

Si tratta di un applicativo "M*ultithread*" che esegue dei cicli di lavoro, che iniziano con un utilizzo intensivo della memoria e finiscono con un utilizzo intensivo della CPU. Il programma distribuisce equamente il lavoro tra un certo numero di thread sincronizzati ed evidenzia il diverso comportamento della JVM in termini di "*tempo di esecuzione*" (ms), man mano che si passa da operazioni totalmente "memory-intensive", ad operazioni totalmente "cpu-intensive". Inoltre, viene evidenziato quanto giova alla macchina, in termini di tempi di esecuzione, distribuire il lavoro tra un numero maggiore di thread.

Input:

Per effettuare il Test è stato necessario aumentare il carico di lavoro in termini di occupazione di memoria e di computazione, per rendere significativi i risultati sulla nostra macchina, rispetto al codice originario.

Il Test è stato eseguito con 5 thread e 2 cicli di lavoro per ogni tipologia di operazioni ("*memory-intensive*" e "*cpu-intensive*"). Vale a dire, che la prima volta vengono eseguiti due cicli di sola "allocazione di memoria", la seconda volta viene eseguito un ciclo di "allocazione di memoria" ed un ciclo di "computazione" e la terza volta vengono eseguiti due cicli di sola "computazione".

Output:

Si sono effettauti 5 tipi di Test, legati all'utilizzo o meno di opzioni a run-time nell'eseguire il comando Java e alla definizione o meno di particolari "variabili logiche". Per ognuna di queste ottimizzazioni, si sono ottenuti risultati differenti:

I° Caso: Il comando "java" è stato dato senza ottimizzazioni

```
$ java HeapTest 5 2 Test.txt
```

ed ha prodotto il seguente risultato:

# Threads	*2 Heap, 0 CPU*	*1 Heap, 1 CPU*	*0 Heap, 2 CPU*
1	12543	17621	21286
2	7849	9943	11293
3	9529	10945	11183
4	10027	10415	11586
5	12835	11801	11200

I dati contenuti nella tabella evidenziano come al crescere del numero dei thread, aumenti il tempo relativo all'utilizzo della memoria (causato dagli accessi concorrenti dei thread) e come diminuisca invece, il tempo relativo all'esecuzione delle computazioni (causato dall'esecuzione "in parallelo" da parte dei thread).

II° Caso: Il comando "java" è stato dato settando un valore iniziale per l'allocazione della memoria

```
$ java -Xms256m 5 2 Test.txt
```

ed ha prodotto il seguente risultato:

# Threads	*2 Heap, 0 CPU*	*1 Heap, 1 CPU*	*0 Heap, 2 CPU*
1	12909	17916	21978
2	8198	9998	11427
3	9437	10419	11051
4	10014	11223	11516
5	11850	11707	11319

I dati contenuti nella tabella evidenziano che le performance di un applicativo che fa un uso massiccio della memoria, migliorano se viene configurata l'allocazione iniziale "garantita" di una certa quantità di memoria.

III Caso: Il comando "java" è stato dato utilizzando l'ottimizzazione del compilatore "JIT" (Just-In-Time)

```
$ java -Xoptimize HeapTest 5 2 Test.txt
```

ed ha prodotto il seguente risultato:

# Threads	*2 Heap, 0 CPU*	*1 Heap, 1 CPU*	*0 Heap, 2 CPU*
1	13387	17756	21921
2	8437	9725	11395
3	9357	10626	11135
4	10136	11046	10752
5	11925	12060	11500

I dati contenuti nella tabella evidenziano che le performance di un applicativo che esegue computazioni in modo massiccio, migliorano se viene configurato l'utilizzo dell'ottimizzazione del "JIT compiler".

IV° Caso: il comando "java" è stato dato settando un valore iniziale per il "new generation space" dell'Heap (o nursery o eden space o young space) e per per la percentuale di questo heap che sopravvive alla "Garbage Collection"

```
$ java -XX:NewSize=512m -XX:SurvivorRatio=2 HeapTest 5 2 Test.txt
```

ed ha prodotto il seguente risultato:

# Threads	*2 Heap, 0 CPU*	*1 Heap, 1 CPU*	*0 Heap, 2 CPU*
1	12999	17600	21053
2	8086	9204	11370
3	8109	9737	11052
4	7913	9554	11450
5	8433	9267	11316

L'opzione "-XX:NewSize=<size>" specifica lo spazio di memoria ("young space") in cui vengono messi gli oggetti appena creati nello Heap e l'opzione "-XX:SurvivorRatio=<size>" indica la percentuale di questi oggetti che sopravviverà alla "Garbage Collection" e che li sposterà in un'area chiamata "old generation".

I dati contenuti nella tabella evidenziano che le performance di un applicativo che fa uso massiccio della memoria, potrebbero migliorare

V° Caso: Application Performance Tuning (utilizzo delle Variabili Logiche)

Il "RTE" (Run-Time Environment) di Java in ambiente OpenVMS è sensibile alla definizione di certi nomi logici, che possono alterare il suo comportamento.

1. La definizione del nome logico "JAVA$FORK_PIPE_STYLE" :

```
$       DEFINE JAVA$FORK_PIPE_STYLE       2
```

ha prodotto il seguente risultato nell'eseguire il Test:

# Threads	*2 Heap, 0 CPU*	*1 Heap, 1 CPU*	*0 Heap, 2 CPU*
1	12869	17628	21650
2	8004	10294	10419
3	8851	10253	10967
4	10510	11046	11112
5	12473	11139	10860

I dati contenuti nella tabella evidenziano che gli applicativi che utilizzano i "thread" e che eseguono computazioni in modo massiccio, sono quelli che beneficiano maggiormente della definizione del nome logico "JAVA$FORK_PIPE_STYLE". Tale nome logico utilizza un'area di memoria (una parte dell'Heap) come buffer, per "appoggiare" momentaneamente le informazioni che i thread potrebbero scambiarsi. In particolare, non utilizza il "mailbox" che è il defualt, ma utilizza il "socket", che richiede uno stack TCP/IP.

Conclusioni:

Nei grafici seguenti vengono riassunti i cinque tipi di test, rispettivamente ai due casi estremi, cioè alle operazioni totalmente "*heap-intensive*" e "*cpu-intensive*":

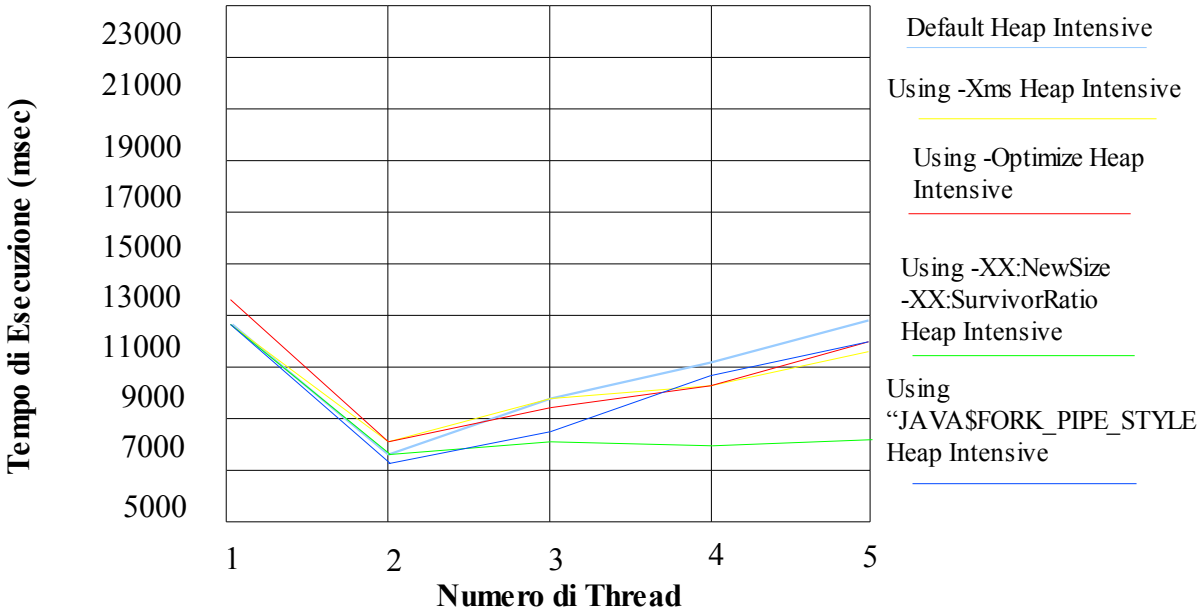


Fig. 1 Grafico riassuntivo per le operazioni totalmente "Heap-Intensive

Il grafico sopra evidenzia come per applicazioni che facciano uso massiccio della memoria, può essere vantaggioso assegnare subito una quantità minima di memoria. Comunque, il maggior vantaggio per questo tipo di operazioni, si ottiene sfruttando a pieno il "Generational Garbage Collection".

Il grafico sotto invece, evidenzia come per applicazioni che eseguano quantità massicce di calcoli, può essere vantaggioso l'utilizzo dell'ottimizzazione del "JIT" compiler. Comunque, il maggiore vantaggio per questo tipo di operazioni è rappresentato dal multithreading ed in particolare, dall'utilizzo da parte di questo, della tecnica del "socket buffer", dove "appoggiare" le informazioni che i thread si scambiano.

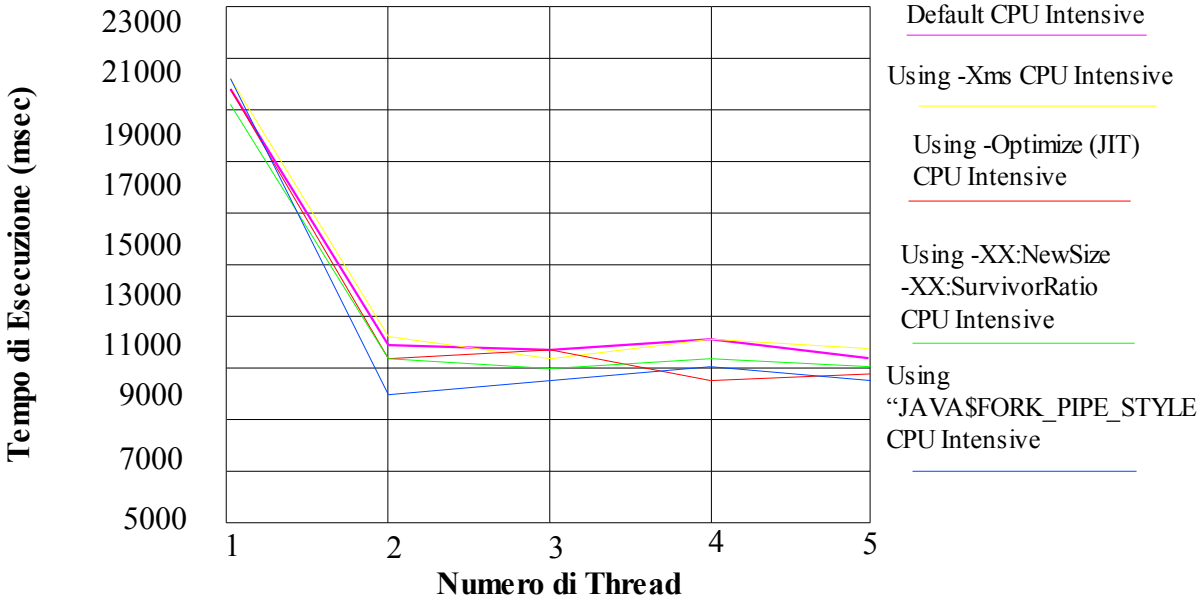


Fig. 2 Grafico riassuntivo per le operazioni totalmente "CPU-Intensive

Inoltre, viene messo in evidenza come l'utilizzo dei thread può essere vantaggioso in applicazioni che eseguano una grande quantità di calcoli, ma che d'altro canto, possono causare troppe contese negli accessi concorrenti alla memoria condivisa.

5.3 Verifica del comportamento della JVM in relazione ad operazioni di I/O su socket e file

In questo paragrafo viene descritto l'utilizzo di un applicativo di tipo "Client/Server", per testare il comportamento della JVM in relazione all'I/O su socket e su file.

5.3.1 Test Program

In questo applicativo, implementato da noi "*ad-hoc*", viene generato un "ServerSocket", il quale si mette in ascolto dei vari Client che vogliono connettersi al Server, per chiedergli di eseguire un'operazione. Il Server è multithread, quindi può ascoltare e soddisfare le richieste di più client contemporaneamente.

Il Test è stato effettuato sulla stessa macchina, per la parte Server e per quella Client, grazie alla connessione in "loopback", specificando la stringa "localhost".
Tuttavia, anche se client e server girano sulla stessa macchina, utilizzano due "console" differenti e di conseguenza, due spazi di indirizzamento diversi. In pratica, si tratta di una valida simulazione di una comunicazione client-server reale.

Una tipica interazione tra client e server, è la seguente:

```
> java MultithreadServer
In attesa di connessione...
```

Sul client:

```
> java MultithreadClient localhost
```

Anche per il Client, si è deciso di avvalersi dei "thread" per lanciare contemporaneamente più istanze di richieste di connessione al Server, per vedere come quest'ultimo, si comporta nella gestione "simulata" di connessioni e richieste multiple.

Il compito che viene richiesto al Server da parte dei Client è semplicemente la traduzione di un "testo". Pertanto, il Client si connette, invia lo "stream" di output al "Server", il quale lo riceve sullo "stream" di input, lo traduce e lo invia di nuovo al Client.
Le operazioni sui socket e sui file, eseguite dal Server, sono piuttosto semplici, tuttavia sono sufficienti per mettere in evidenza quali tipo di ottimizzazioni possono essere effettuate per aumentare le sue "performance".

Input:

I codici Server e Client sono stati eseguiti sulla stessa macchina, utilizzando però, due console differenti. Per il lato Client, per simulare l'accesso al Server da parte di più utenti

contemporaneamente, si sono utilizzati i thread, come d'altro canto, si sono utilizzati a maggior ragione, implementare il Server.

Il Client che si connetterà al Server, gli chiederà se ha già disponibile la traduzione di un file, in caso di risposta negativa da parte del Server, il Client gli invierà il file da tradurre, il Server ne farà una copia, ne farà la traduzione e poi la invierà al Client, come richiesto.

Il Test è stato eseguito simulando 10 richieste di connessioni "simultanee" al Server, da parte di altrettanti Client.
L'obiettivo è quello di misurare i tempi di risposta del Server alle richieste dei Client e valutare se è possibile apportare delle ottimizzazioni, agendo direttamente sulla "Java Virtual Machine" per mezzo di opzioni e variabili logiche.

Output:

Si sono effettuati 4 tipi di Test, ciascuno legato alla definizione o meno di particolari variabili logiche. Per ognuna di queste ottimizzazioni, si sono ottenuti risultati differenti:

I° Caso: Il comando "java" è stato dato senza ottimizzazioni

Si sono ottenuti i seguenti risultati:

Client	***Tempo di Risposta (msec)***
7	10774
9	11892
6	14315
2	14314
3	14325
10	14521
8	14633
4	14711
1	14750
5	14723
Media	***13896***

La prima colonna indica l'ordine con cui il Server ha risposto ai Client e la seconda indica il tempo totale in cui il Client ha ottenuto il file tradotto dal Server.

II° Caso: Utilizzo della variabile logica "java$caching_interval"

Settando questa variabile logica con un valore, che rappresenta il tempo, si permette all'applicazione di tenere in memoria, per il tempo specificato, il risultato delle operazioni di verifica sui file, come ad esempio, ricordarsi che un file è stato appena creato, o appena testato per la sua esistenza, ecc....

```
$    DEFINE/JOB    JAVA$CACHING_INTERVAL    60
```

Pertanto, la nostra applicazione, per i prossimi 60 secondi, troverà in "cache", il risultato dei test sull'esistenza dei file che gli sono necessari e pertanto, non li rieseguirà per i 60 successivi secondi. Su un'applicazione che fa un uso massiccio di operazioni di verifica sui file, ne può derivare un considerevole risparmio di tempo. Infatti, già nella nostra applicazione, è possibile notare un certo risparmio di tempo:

Client	***Tempo di Risposta (msec)***
8	10882
4	13091
6	13548
9	13385
1	13419
7	13296
10	13620
3	13391
5	13880
2	14304
Media	***13281***

III° Caso: Utilizzo della variabile logica "JAVA$TIMED_READ_USE_QIO"

Settando questa variabile logica, si riesce a diminuire il tempo che l'applicazione spenderebbe in operazioni di tipo "kernel mode" a discapito di quelle "user mode", come ad esempio, può accadere nel controllo del time-out nei "socket stream". Pertanto, le applicazioni che maggiormente possono beneficiare di questa variabile logica, sono quelle Web, quelle che eseguono servizi TCP/IP, quelle che utilizzano i socket. Anche la nostra applicazione ha trovato un certo giovamento nell'utilizzo di questa variabile, rispetto al caso senza ottimizzazioni:

Client	***Tempo di Risposta (msec)***
9	10474
6	13062

2	13227
1	13343
7	13397
5	13587
3	14282
8	14276
4	14455
10	14270
Media	***13437***

IV° Caso: Utilizzo della variabile logica "DECC$FILE_SHARING"

Molte applicazioni Java si aspettano l'utilizzo da parte del "Run-Time" della modalità "file-sharing" tipica dei sistemi UNIX. Anche nei sistemi OpenVMS può essere ottenuto lo stesso risultato, utilizzando la variabile logica "DECC$FILE_SHARING". Settando questa variabile logica si sceglie di privilegiare le performance rispetto alla sincronizzazione in lettura/scrittura con il disco:

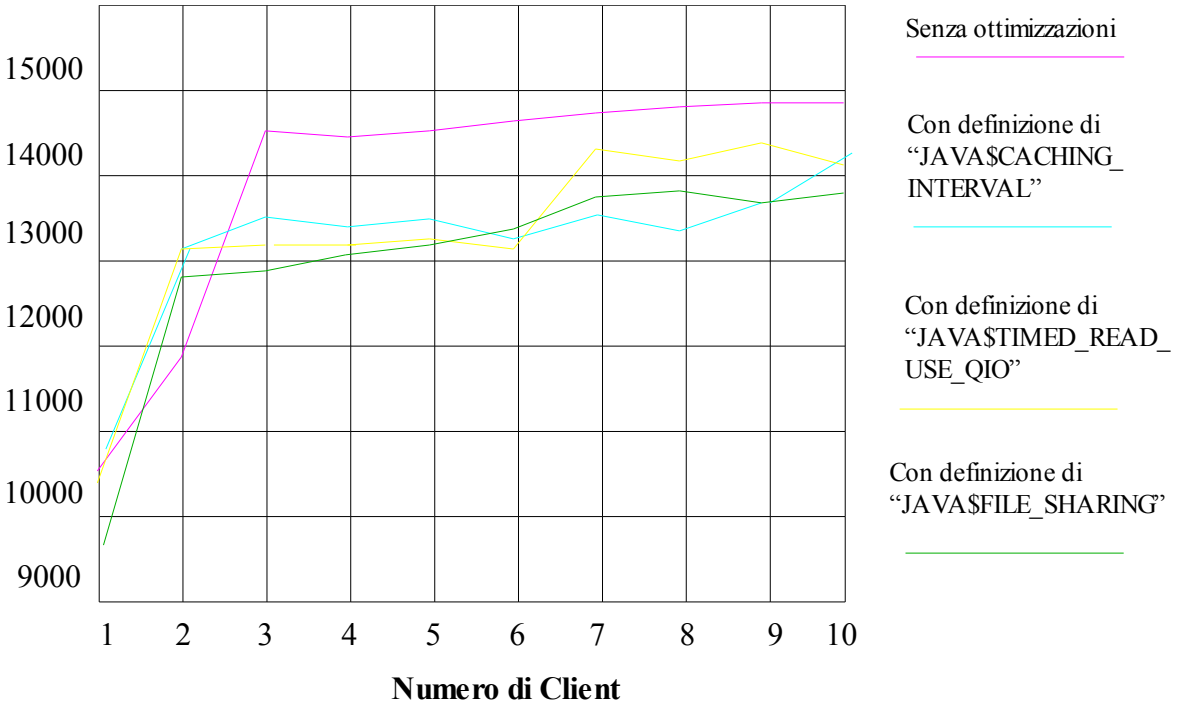
```
$ DEFINE DECC$FILE_SHARING ENABLE
```

La nostra applicazione ha ottenuto un certo risparmio di tempo rispetto al caso in cui la variabile logica non è abilitata:

Client	***Tempo di Risposta (msec)***
9	9548
6	12816
5	12861
8	13140
7	13364
3	13787
1	13816
4	13899
2	13868
10	13891
Media	***13099***

Conclusioni:

Nel grafico seguente vengono riassunti i quattro tipi di test effettuati per valutare il comportamento della JVM in presenza di operazioni su socket e file:



Dal grafico si può notare come al crescere del numero dei client, aumenti ovviamente, anche il tempo di risposta del server e come il maggior vantaggio per i client si ottenga quando sul server è abilitata la variabile logica "DECC$FILE_SHARING", tipica del mondo UNIX, che privilegia le performance rispetto alla sincronizzazione in lettura/scrittura con il disco.

5.4 L'applicativo JBoss

JBoss è un Application Server "*J2EE ™ compliant*", vale a dire che rispetta le specifiche stabilite dalla Sun™ riguardo allo svilluppo di applicazioni "*Java 2 Enterprise Edition*".

JBoss è un applicativo implementato completamente in Java e pertanto, tutte le valutazioni effettuate sulla gestione della memoria, sulla "*Generational Garbage Collection*" ed sul settaggio dei "*nomi logici*" per l'ambiente Java, possono essere applicate anche a JBoss, in relazione alla tipologia dei servizi che si intendono sviluppare ed eseguire sull'Application Server, come ad esempio, web services, applicativi di tipo enterprise, EJB, portali, etc.

In particolare, sulla nostra installazione, sono state testate due tipologie di servizi offerti da JBoss, il "*Java Message Service*" (JMS) e l"*Enterprise Java Bean*", attraverso due semplici applicativi, scaricabili dai seguenti siti:

http://www.javaportal.it/rw/19651/13381/25848/23580/editorial.html

http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossSimpleExample